SDSS Virtual Data Requirements: A Weak Lensing Map as a Prototype Virtual Data Set

James Annis[1], Steve Kent[1], Alex Szalay[2]

[1]Experimental Astrophysics Group, Fermilab
[2]Department of Physics and Astronomy, The Johns Hopkins University

Draft v0 April 6, 2001

We examine the Sloan Digital Sky Survey data pipelines in the light of virtual data. After a brief review of the SDSS project, pipelines, and data products, we consider the intersection of GriPhyN interests and SDSS activities. We then suggest that the appropriate prototype virtual data set is a weak lensing map derived from the co-added Southern Survey of the SDSS (where coadd is the process of adding images together). The request for the weak lensing map is a request for a catalog made from the coadded images, the existence or non-existence of which determines the need for running all of the SDSS pipelines, running special coadding code, and re-running the SDSS pipelines. While astronomers naturally focus on the image processing step of the coadd, the bulk of the process is dealing with the same multi-faceted bookkeeping problem that faces all Virtual Data applications. The creation of the coadded South is a challenge that the SDSS collaboration will be solving over the next two years.

1 Introduction

The analysis of weak lensing signals provides one of the very few direct measurements of mass available to astronomy, and there is great interest in pursing the analysis to the faintest magnitudes and widest sky coverage available. While the main SDSS survey does not reach limiting fluxes faint enough to pursue true weak lensing maps, the SDSS Southern Survey does. In this survey, the same area of sky is imaged 20-60 times and the images added together to produce a image that reaches sqrt(N) times deeper in flux. The sqrt(N) is of interest: while the full complement of imaging data provides the maximal signal to noise, much of the improvement comes in the first handful of images and thus intermediate coadds are of interest.

We take the challenge of building weak lensing maps as the prototype Virtual Data problem in the Sloan Digital Sky Survey.

1.1 Description of the SDSS

The Sloan Digital Sky Survey is a project to produce a map the Northern Celestial sky. We use a 150 Megapixel camera to obtain 5-bandpass images of the sky; the final 5 images are each 1 million by 1 million pixels. We then target the 1.2 million brightest astrophysically interesting for spectroscopy, allowing us to produce 3-dimensional maps of the galaxies in the universe out to 3 billion light years, luminous elliptical galaxies out to 5 billion light years, and quasars out to 10 billion light years. If the spectra were combined row by row into an image, they themselves would be an image of 4096 pixels by 1.2 million pixels.

The survey is performed at Apache Point Observatory in New Mexico, where the specially designed wide field 2.5m telescope is sited, along with its 0.5m photometric telescope and with instruments built by Princeton and Johns Hopkins University. The data are reduced at Fermilab using pipelines constructed by Princeton, University of Chicago, The US Naval Observatory, Fermilab, and the Johns Hopkins University.

1.2 Description of SDSS data products

There are a variety of data products from the SDSS.

catalogs	measured parameters of all objects	500 Gig
atlas images	cutouts around all objects	700 Gig
binned sky	sky leftover after cutouts, binned 4x4	350 Gig
masks	regions of the sky not analyzed	350 Gig
calibration	a variety of calibration information	150 Gig
frames	corrected images	9.2 Terabyte

The Southern Survey, taken run by run amasses to 2/3rds of those totals.

The components of any given 200 sq-degree coadd of the south come to:

catalogs	measured parameters of all objects	10 Gig
atlas images	cutouts around all objects	15 Gig
binned sky	sky leftover after cutouts, binned 4x4	7 Gig
masks	regions of the sky not analyzed	7 Gig
calibration	a variety of calibration information	3 Gig
frames	corrected images	Nx200 Gig

1.3 Description of SDSS Pipelines

There are a large number of pipelines that run more or less sequentially on theSDSS data.

Pipeline	**Function**
SSC	preliminary astrometric information
Astrom	calculate the astrometric calibration
Postage Stamp	determine the point spread function
Photo	measure objects in the frames
Mtpipe	calculate the photometric calibration
Nfcalib	apply the photometric and astrometric calibrations
Opdb	store the data used in operations
Target	select targets for spectroscopy from the data
Tile	optimize the placement of the spectroscopic plates
Plate	design the plates: take sky coordinates down to machine drilling x,y
Spectro-2d	extract the 600 fiber spectra from the 4 spectroscopic images
Spectro-1d	measure the redshift and classification of the spectra
SX	the catalog science database
Atlas-DB	the atlas image/spectra science database

All of these pipelines are glued together by a over-pipeline called DP whose sole job it is to get the outputs from one pipeline into the next pipeline. The interfaces to these pipelines are controlled by a data model held under change control; this is necessary but not sufficient.

2 GriPhyN and the SDSS

A big challenge for GriPhyN is to capture as much as possible the ability to generate every possible reduced data set, presuming that only the raw data are maintained permanently. This is indeed a big challenge, particularly for the experiments, where data processing pipelines are incompletely automated. However, it is an interesting exercise to see if one could describe the steps in SDSS data processing in a way that is generic, modular, and could be exercised in a framework envisioned by

GriPhyN. Indeed, a major challenge of GriPhyN is to define a model for data processing that is tractable for experiments to implement, general enough to meet the needs of a diverse set of experiments, and useful enough to be desirable in a distributed "grid" environment.

The SDSS data processing is file oriented, as are the LIGO examples that we have seen. Atlas/CMS are object oriented. Certain SDSS queries will also be object oriented (given an ID for an object on the sky, give me a full set of data) - however, the object ID can always be mapped to a file that contains the object, so the translation is easy. I think of these as data requests. More interesting situation will involve queries against attributes that may not exist - e.g., return data for an object nearest a given position on the sky. This cannot be done if there is no catalog of objects in the first place; instead, there must be a mechanism to convert the query over objects into a query on attributes of request-able "objects" (which, in the SDSS case, are mapped into all files that such objects might reside in).

Data processing in SDSS is procedure oriented: start with raw data, create processing jobs (multiple stages), output files are created at the end. For a GriPhyN environment, we really want to go the other direction: given a file, say, what is needed to created it? This requires thinking of the data processing flow in "reverse order".

2.1 Description of the Abstracted SDSS Pipeline

Since data processing consists of running a series of pipelines, we wish to characterize each pipeline in the same fashion, so that the totality of processing is the combination of a number of modular steps. In SDSS land, we call this the "factory".

A single pipeline has inputs and outputs of the following flavors:

INPUTS:

1. A plan file that defines which set of data are to be processed, the root directories for input & output files.
2. One or more parameter files - tuning parameters applicable to this particular set of data. Often there is a default set of parameters provided with the pipeline code. Occasionally these are overridden to handle special properties of the data.
3. Various input files, that are the products of upstream pipelines. The pipelines usually have internal knowledge of how such files are named and located relative to the root directories.
4. Environment variables. These are used to define which versions of pipelines are being run. A single pipeline may have multiple dependent products. Normally, though, the version of a pipeline product is uniquely related to the versions of all dependent products.

OUTPUTS:

1. Output files that are the data products themselves
2. Log and error files
3. Quality control files. These identify any outputs that are

flawed such that subsequent pipelines cannot run. They do not identify outputs that are out of bounds regarding a quality threshold but do not impede the running of a follow-on pipeline.
4. A single status flag. 0 = next pipeline can run, 1 = hand intervention required.

A complete "job" consists of the following steps:

1. Preparation. Stage any inputs that are needed. Create plan files, usually based on some rules about where directories are located.
   Make sure disk space is available. Inputs needed from previous pipelines are checked by looking for a "QC" file produced by any upstream pipelines.
2. Submit the pipeline job. The pipeline usually receives a few key inputs, such as the location of plan files. Return 0 or 1.
3. Run a status verify job that checks if the submitted job did complete.
4. Run a pipeline verify job to generate QC information and starts the follow-on pipeline. Return 0 or 1.
5. Run a "scrub" job that removes most files once they have been archived. The archiving is itself a "pipeline"

What would be required to invert the above procedure - i.e., given a request for a particular output file, what additional work is needed to trigger the required processing? This is an area of a bit of research.

2.2 The Factory Mode

The factory itself is the DP scripts that join together the pipelines. There are 3 generic stages of the factory at each and every pipeline:

1. Prep
   - generate plan file
   - make relevant directories
   - make sure the relevant data is available
   - make relevant sym links
   - locate space
   - register the resources reserved in a flat file database
   - Call submit

2. Submit
   - generate a shell script the fires off the batch system submit
   - Call ender

3. Ender
   - periodically check status of a pipeline submit by looking for
     the files that should be created. The existence of the
     files is necessary and almost sufficient for the next step
     to succeed.
   - after completion of the pipeline, run a quality control script,
     where various quantities (e.g., the number of galaxies/image)
     are given a sanity check. if QC checks, then

Call Prep for the next pipeline.

These are daisy chained: the first invocation of Prep takes as an argument how many of the following pipelines to run in series.

Having said all of this, it is worth pointing out just how many control variables there are for each pipeline step. We'll use as an example the plan file and parameter file for SSC, a simple pipeline. The plan (see appendix A) is straightforward and can be script generated. The parameter file (see appendix B) contains the parameters that control the behavior of the pipeline. It is entirely conceivable that any of these may wish to be changed by someone invoking the pipeline (say, to invoke a stray light correction necessary for some but not all of the data); in practice these don't change very often.

2.3 GriPhyN and the SDSS

It is worth noting that the SDSS factory is up and running. GriPhyN will not have an influence on the design of either the pipelines or factory scripts, nor are the tools developed likely to make it into the normal production modes. Having said that, the Southern Survey provides an opportunity for GriPhyN to have an impact, and to learn from a very short term problem that requires success, not designs. Furthermore, we are, unlike CMS, more interested in the virtual data technologies than the data grid technologies, for the simple reason that the data reduction happens at a single site with sufficient, though distributed, computing resources and the resulting data is small enough to push around on the current Internet.

GriPhyN can contribute a framework for dealing with our factory in a request driven mode, as opposed to our current data arrival driven mode: this may well take the form of learning how to give a concrete and useful specification for deriving the data. GriPhyn can contribute mechanisms for dealing with multiple copies of the same data, both as input and output. We believe there is an opportunity for GriPhyN to contribute compute resource discovery and reservation systems: Virtual Data systems require coordinated computation and data management.

The very first step might well be to take our existing infrastructure and describe in terms of the Data Grid Reference Architecture.

3 The SDSS Virtual Data Problem: The Southern Coadd

We take as the virtual data driver the science of analyzing the weak lensing map of the Southern coadded data. First things first: the southern coadd must be created, and that is only possible using existing data.

```
1) The Southern Survey Coadd
        run full pipelines on incoming data
        given an area on which to coadd,
                find relevant reduced data
                find disk space and compute power
                extract relevant reduced data from long term storage
                build mapping function from calibration data
```

perform coadd to create new reduced data
keep track of intermediate coadd catalogs and data

2) Run weak lensing analysis on the intermediate coadded south
    - locate atlas images on disk
    - compute optimal shape parameter
    - produce shape catalog

Clearly the right approach to the existing SDSS factory is to place it as a single coherent entity into the GriPhyN framework, as opposed to trying to put the individual pipelines into the framework.

Solving this problem provides avenues for future progress. One vision of the SDSS/NVO team is to spin the SDSS data on a compute cluster and allow for demand driven re-reduction with the ever-improving versions of Photo. Of course, one cannot just run Photo, as it is the center of a long processing chain, but it is exactly this processing chain that must be made demand driven to solve the weak lensing map problem.

3.1 The Southern Survey Coadd In Design Detail

1. given an area on which to coadd,
    a. Take user input
2. find relevant reduced data
    a. Query either a db or a flat file for the the runs that have been observed and extract the list of runs that are of the right strip.
    b. Using that list, determine which runs overlap the given area. Involves the calculation of a spatial intersection.
    c. Apply cuts against data that are not of high enough quality to use (even though other parts of the run may be.)
    d. Estimate how much data is involved.
    Metadata: a list of runs and portions of runs involved

3. find disk space and compute power for input data, processing, and outputs
    a. Query the local network for available machines
    b. Query the local network for available storage
    c. Reserve the resources for a period of time
    Metadata: a list of machines

4. extract relevant reduced data from storage
    a. From some knowledge base, determine for each run if the data is:
        1) on archival disk
        2) on production disk
        3) on fast tape
        4) on slow tape
    b. Arrange to get the data off the media and onto the production machines

Metadata: a list of portions of runs and where they live on production disk

5. build mapping function from calibration data
    - a. An image is a 1361x2048 array of pixels. In general two images of the "same" piece of sky will be:
        - 1) slightly offset in both x and y
        - 2) slightly rotated
        - 3) have different small distortions as a function of x,y superimposed
        - 4) have different conversion between pixel value and energy flux
    - b. These translations, rotations, distortions, and scalings are calculateable from a set of calibration information that may or may not be kept as a header to the data itself.
    - c. Find the calibrations, and build the mapping function for each pixel

    Metadata: computed mapping functions to be applied to the data

6. perform coadd to create new reduced data
    - a. Load a set of co-located images into memory
    - b. Apply mapping function
    - c. Perform median or average on stack of pixels (in truth: apply very clever algorithm to make maximal use of information and minimize noise)
    - d. Save output to disk.

    Metadata: Location of output data

7. run full SDSS pipeline on new data set
    - a. arrange for all necessary input files to be in place
    - b. arrange for all 10 pipelines to run
    - c. watch success or failure of the pipelines
    - d. save resulting outputs to disk

    Metadata: Location of output data

8. keep track of intermediate coadd catalogs and data
    - a. The output of 6. is to be preserved, as 7. can be done multiple times
    - b. The output of 7. is to be preserved
    - c. Each time new data comes in, the above is done.

9) Run weak lensing analysis on the intermediate coadded south
    - a) locate atlas images on disk
    - b) compute optimal shape parameter
    - c) produce shape catalog

4 Conclusion

The Southern Survey Coadd presents a local, manageable version of the full Virtual Data problem, and one which exhibits most of the major challenges of the full problem. Our collaboration will be ramping up to perform the Southern Survey over the next two years, and one way or another we will surmount the challenges and produce the weak lensing map. However, our activities during the process present the GriPhyN collaboration with an opportunity for alpha testing of tools, techniques, and ideas.

5 Appendix A: An Example SDSS Pipeline Plan File

A plan file is essentially the parameters controlling a batch job submission. The plan file presented here is that from the SSC, the first of the SDSS production pipelines and among the most straightforward. Production of these files could clearly be under the control of GriPhyN software.

```
action          ssc
version          "v4_6"
parameters        scParam.par
report          idReport-51075.par

run 94
rerun 5
camCol 1
endCol 1
startCol 1

startField 11
endField 546

koVersion "v2_0"
verbose 0
display 0

ccdrow_r       1
ccdrow_i       2
ccdrow_u        3
ccdrow_z       4
ccdrow_g       5
ccdrow_o        6
ccdrow_l       7
ccdrow_t       8
ccdrow_s       9

catalog_type    equatorial

aGangInputDir       /data/dp3.b/data/94/5/gangs
pGangInputDir       /data/dp3.b/data/94/5/gangs
fieldInputDir      /data/dp3.b/data/94/5/fields/1


fieldInputDevice    disk_photo
fieldOutputDevice   disk_photo
```

```
tapeDeviceName      disk
tapeLabel           NULL
tapeLabelLength     80
tapeSkipFiles       0
tapeLogFile     /data/dp3.b/data/94/5/logs/idTapeLog-NULL.par
writePhotomOnly     0
ignorePhotomData    0


inputDir /data/dp3.b/data/94/5/ssc
parametersDir       .
keepPhotomImages      1
diagnostics          1
diagFile         scDiag
koDir /data/dp3.b/data/94/5/ssc
koFileBase koCat
wingFile         scWing
frameFile        scFrame
configDir /data/dp3.b/data/94/5/logs
ccdConfig opConfig-51437.par
ccdECalib opECalib-51371.par
ccdBC          opBC-51008.par
ccdCamera opCamera-51075.par

outputDir /data/dp3.b/data/94/5/fangs/1
outputFangDir /data/dp3.b/data/94/5/fangs/1

QAplots 1
```

6 Appendix B: An Example SDSS Pipeline Parameter File

A parameter file contains the parameters that an astronomer might wish to change upon running the pipeline. The one presented here is from the SSC pipeline, the first of the SDS production pipelines and among the most straightforward. It is worth noting that in princple the requester of Virtual Data might wish to change any or all of these parameters for any or all of the pipelines; indeed it is this possibility that is among the drivers of the need for Virtual Data. The problem is that then all of these parameters must be presented to the Virtual Data requestor for possible change, along with sufficient documentation for them to understand what they are changing.

```
#
# Software parameters for SSC
#

# bad chips
badCCDs {76}

# frame size in pixels
vertical_frame_size   1361

# matching stars
am_xpos_field rowCentroid     # name of field (from SCPARAMBS structure)
                              # used to compare astrometric stars in one
                              # direction
am_ypos_field colCentroid     # name of field (from SCPARAMBS structure) used
                              # to compare astrometric stars in other
                              # direction
am_match_diff40               # two astrometric stars with centers which are
                              # this close (or closer), in pixels in each
                              # direction (not in true geometric distance)
                              # are considered a match
ignore_DaCentroid_errors 1    # ignore errors from atDaCentroidFind?
#
# defining flat-fielding method
#
ff_none             0              # perform no bias-subtraction or flatfielding
ff_quartiles   1              # use a flatfield vector based on the quartiles
ff_fixed       0              # use a fixed flatfield vector in a disk file
ff_filename    scFlat_        # base of the filename inside which the
                              # flatfield vector for each chip lives
                              # Ex:  scFlat_42.fit, 42=camera row and column
ff_maxncol     2128           # max number of cols in any photometric chip
```

```
#
# discarding bad data
#
bc_distance     10                # if a star is this close (pixels) to
                                  # a known chip defect, ignore it
#
# calculating sky
#
sk_edge_npix  10                  # use pixels within this many of edge
                                  # of a postage stamp to calc sky
sk_clip_nsigma3.0                 # clip values this many sigma from mean
                                  # when calculating sky value in stamp
sk_clip_niter   2                 # make this many iterations in clipping
                                  # when calculating sky value in stamp
st_recenter_boxsize     25        # if > 0, calc center of a star using
                                  # first moments inside a box with this
                                  # half-length at center of stamp, and
                                  # re-cut stamp at the new position.
                                  # if == 0, do not re-cut stamp; keep stamp
                                  # based on position in the astrometric chips
st_object_threshold   8.0      # minimum peak level (in sky sigma) for an object
                            # to be added to the stamp list
st_object_nmax         100      # maximum number of additional object per frame
st_conv_sigma 1.2                 # width (pix) of gaussian with which to
                                  # convolve stamps
#
# How to clip the data
#
ds_statusfit_delete       0       # if 1, discard stars which have "statusfit"
                                  # field != 0
ds_filterlist        g            # only test these filters for valid parameters
ds_minrmajor  0.4                 # rMajor must be > this to be OK
ds_maxrmajor   15.0               # rMajor must be < this to be OK
ds_minrminor  0.1                 # rMinor must be > this to be OK
ds_maxrminor   10.0               # rMinor must be < this to be OK
ds_minpeak      1.0               # peak must be > this
ds_maxpeak  66000.0               # peak must be < this
ds_radius_bit     1               # set this bit in the "sscstatus"
                                  # field of SCPARAMBS if any radius
                                  # length is outside allowed range
ds_radius_delete  0               # if 1, discard stars which fail radius tests
ds_peak_bit       2               # set this bit in the "sscstatus" field of
                                  # SCPARAMBS if peak is outside allowed range
ds_peak_delete  0                 # if 1, discard stars which fail peak tests

#
# "wing" stars parameters
#
```

```
ws_pix_offset  400          # max number of pixels away from center
                            # of its CCD chip for a "wing star"
ws_min_mag     8.0          # minimum mag allowed for "wing star"
                            # (in whatever passband is in catalog)
ws_max_mag     14.0         # maximum mag allowed for "wing star"
ws_box_size    200          # size of box, in pixels, which we cut

#
# "frame" stars parameters
#
fs_pix_offset  300          # max number of pixels away from center
                            # of its CCD chip for a "frame star"
fs_min_mag     1.0          # minimum mag allowed for "frame star"
                            # (in whatever passband is in catalog)
fs_max_mag     8.0          # maximum mag allowed for "frame star"
```